Patryk Ostrowski
Nr. Indeksu: 240671
Prowadzący: Marta Emirsajłow
Grupa: Piątek 7:30

Projektowanie algorytmów i metody sztucznej inteligencji

Projekt gry w warcaby z wykorzystanie algorytmu minmax

1 Wstęp

Algorytm MinMax służy w celu zminimalizowania możliwych strat oraz zmaksymalizowania możliwego zysku. Na podstawie tego algorytmu można opracować sztuczną inteligencje do wielu gier. Zdecydowałem się pokazać wykorzystanie jego w warcabach.

2 Opis tworzonej gry

W warcaby klasyczne gra się na planszy 8 na 8 pól. Każde pole może być: puste, zawierać jeden pion lub damke (w obu przypadkach o różnych kolorach). Otrzymujemy więc 5 możliwych stanów dla każdego pola, dlatego do stworzenia planszy wykorzystałem tablicę typów wyliczeniowych. Klasa 'Board' posiada poza tym metody umożliwiające wygenerowanie wszystkich możliwych ruchów dla danego piona lub całego koloru, sprawdzenie czy występuje możliwość bicia dla piona / koloru, wykonanie ruchu, funkcję oceniającą sytuacje na planszy oraz funkcję sprawdzającą stan gry.

Do przechowania ruchów zaimplementowałem dwie struktury: Coords, zawierającą współrzędne piona przed i po wykonaniu ruchu oraz Move, zawierającą wszystkie koordynaty dla danego ruchu (kiedy składa się na przykład z 3 bić).

Graficzna reprezentacja gry została wykonana przy użyciu biblioteki SFML

Przyjęte zasady:

- Bicie jest obowiązkowe
- Piony mogą poruszać się tylko do przodu, ale dozwolone jest bicie wstecz
- Jeśli po biciu można wykonać kolejne, to również jest ono obowiązkowe
- Nie ma obowiązku wybrania najdłuższej sekwencji bić
- Jeśli pion dojdzie do ostatniego rzędu, to jest promowany na damkę
- Jeśli pion po wykonaniu bicia znalazł się na polu promującym, ale dalej posiada bicie, to
- promocja nie następuje
- Damka po biciu może przemieszczać się dalej, dopóki nie napotka przeszkody
- Koniec gry następuje kiedy jeden z graczy nie będzie miał możliwości wykonania następnego
- ruchu
- Sytuacja patowa następuje kiedy oboje z graczy wykonają po 15 ruchów damkami bez zmiany
- ilości figur na planszy

3 Wykorzystane techniki AI

3.1 Drzewo możliwych stanów

Dla drzewa przeszukań, zawierającego wszystkie możliwe stany planszy dla n-następnych ruchów, zaimplementowałem klasę pojedyńczego elementu takiego drzewa. W konstruktorze przyjmuje on obiekt planszy, maksymalną głębokość drzewa, kolor, którym gra gracz oraz jako argument opcjonalny przyjmuje on wykonany ostatnio ruch.

Aby utworzyć drzewo należy utworzyć jego korzeń t.j. pierwszy element, który otrzymuje aktualną plansze gry. Do pierwszego elementu nie podajemy jaki ruch został wykonany, ponieważ znajdzie się w nim ruch wybrany przez algorytm Min-max. Konstruktor elementu jest funkcją rekurencyjną,
która wywołuje się, aż do osiągnięcia maksymalnej głębokości drzewa, lub gdy po wykonanym ruchu następuje zakończenie gry.

Konstruktor pobiera z planszy wszystkie możliwe do wykonania dla danego gracza ruchy, następnie dla każdego ruchu tworzy potomka, któremu przekazuje: ruch, obiekt planszy po wykonaniu danego ruchu, oraz kolor pionków, które będą wykonywały ruch następne.

Jeśli zostanie spełniony jeden z warunków zakończenia pogłębiania drzewa, wywołana zostaje funkcja heurystyczna, oceniająca ostateczną sytuacje na planszy.

3.2 Funkcja Heurystyczna

Funkcja heurystyczna pozwala na przybliżenie rozwiązania danego problemu bez zapewnienia gwarancji poprawności uzyskanego wyniku. Używa się jej wtedy, gdy właściwy algorytm jest zbyt złożony, kosztowny lub nieznany.

Od implementacji tej funkcji zależy efektywność całego algorytmu wybierania najlepszych ruchów. Oczywistym czynnikiem jest ilość posiadanych pionów oraz damek. Starałem się punktować dodatkowo piony ’bezpieczne’, t.j. takie, które nie mogą zostać zbite (przylegają do bocznej ściany planszy). Dodatkowe punkty przyznaję też za wyprowadzenie piona z rzędów początkowych,
ponieważ uważam, że piony poruszające się w bardziej zwartej formacji są efektywniejsze. Podstawową mechaniką warcabów jest wykonywanie zbić, dlatego każdy pion z możliwością bicia również
otrzymuje dodatkowe punkty.
Zastosowany system przydzielania punktów za określone czynniki na planszy:

- Wygrana 10000 pkt
- Przegrana -10000 pkt
- Remis 0 pkt
- Każdy pion +7 pkt
- Każda damka +25 pkt
- Każda figura przy bocznej ścianie planszy +2pkt

- Każda figura oddalona o 1 pole od bocznej ściany +1pkt
- Każdy pion w rzędzie dalszym niż 1 +2pkt
- Każdy pion w rzędzie dalszym niż 2 +1pkt
- Każdy pion w rzędzie dalszym niż 3 +1pkt
- Każda figura z możliwością wykonania bicia +15 pkt

3.3 Algorytm MinMax

Algorytm ten jest metodą wybierania ruchu, który pozwala na minimalizację możliwych strat lub maksymalizację zysków. Jeśli dla danej sytuacji na planszy stworzymy drzewo wszystkich możliwych układów dla n-następnych ruchów i każdej przypiszemy wartość przy użyciu określonej funkcji heurystycznej, to możemy wybrać taki ruch, który daje nam największe korzyści. Gracz, dla którego wywołany jest w/w algorytm będzie dążył do maksymalizacji zysków, natomiast przeciwnik będzie się starał te zyski zminimalizować.

Algorytm MinMax dostaje jako parametr korzeń drzewa stanów. Następnie sprawdza, czy aktualny gracz dąży do maksymalizacji, czy do minimalizacji zysków i zależnie od tego wybiera ruch o
największej / najmniejszej wartości. Aby to uzyskać, wywołuje się rekurencyjnie aż do osiągnięcia warunku podstawowego, dzięki któremu może zwrócić konkretną wartość.

Jeśli założymy, że dla każdego ruchu można wykonać n-następnych, to złożoność obliczeniowa algorytmu wyniesie $O(n^m)$, gdzie m to maksymalna głębokość rekurencji. W warcabach liczba możliwych ruchów jest zmienna, więc nie da się dokładnie określić złożoności obliczeniowej algorytmu.

3.4 Cięcia Alfa-Beta

Cięcia αβ nie są osobnym algorytmem, a raczej modyfikacją do algorytmu MinMax. Zakładają one dodanie dwóch zmiennych, które przechowują minimalną β i maksymalną - α wartość, jakie MinMax obecnie może zapewnić na danej głębokości drzewa lub wyżej. Przy starcie algorytmu $\alpha = -\infty$ i $\beta = \infty$, wartości te są korygowane w dalszych etapach działania algorytmu. Na ich podstawie, jeśli przy sprawdzaniu kolejnego poddrzewa węzła minimalizującego $\alpha > \beta$ algorytm może odciąć dane poddrzewo (w tym przypadku cięcie β), ponieważ już wie, że maksymalna wartość tego poddrzewa przekroczy minimalną, którą węzeł może obecnie zagwarantować. Bardzo korzystnie wpływa to na złożoność algorytmu, pozwalając osiągnąć dużo lepszy czas wyszukiwania optymalnego ruchu lub zwiększyć głębokość rekursji. Przy identycznych założeniach, jak wspomniane w opisie algorytmu MinMax, złożoność obliczeniowa algorytmu wynosi $O(n^{m/2})$

4 Podsumowanie i wnioski

Algorytm radzi sobie bardzo dobrze w rozgrywce. Złożoność algorytmu jest dosyć duża, ze względu na ilość ruchów, które algorytm ma do rozpatrzenia. Na moim laptopie gra przestaje być grywalna przy pewnej ilości następnych ruchów. Algorytm nie jest niepokonany, skuteczność jego działania jest uzależniona od tego, jak dokładnie funkcja heurystyczna przybliża dany system, a obecnie zaimplementowana jest z pewnością uboga i daleka od idealnej oceny planszy. Mimo wszystko algorytm sprawia trudności.